Ugo Giordano

Tutor: Prof. Stefano Russo

XXIX Cycle - III year presentation

In-production continuous testing for future Telco Cloud

Università degli Studi di Napoli
Federico II

Short bio

- Master Degree in Computer Engineering
  - **University of Naples Federico II** in 2012.
- Member of the **DEpendable Systems and Software Engineering Research Team** (**DESSERT**)
  - led by Prof. Stefano Russo

- 1-year Fellowship financed by PON Project “SECTOR”
  - Funded by the public-private laboratory COSMIC
- I have worked in applied research project with Finmeccanica Company and Huawei Technologies (a Chinese TELCO equipment Company)

Credit Summary

Student: Ugo Giordano									Tutor: Stefano Russo									Cycle XXIX									
ugo.giordano@unina.it									stefano.russo@unina.it																		
	Credits year 1								Credits year 2									Credits year 3									
		1	2	3	4	5	6			1	2	3	4	5	6			1	2	3	4	5	6				
	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Total	Check
Modules	24	0	0	6	0	3	15	24	13	7	6	0	0	0	0	13	0	0	0	0	0	0	0	0	37	30-70
Seminars	6	0	0	3	0	2,7	2	7,7	8	1,8	3	0	0	0,3	8	13,1	0	0,8	0	0	0	0	0	0,8	22	10-30
Research	35	8	8	2	9	8	2	37	45	5	7	7	8	9	9	45	60	8	9	10	10	10	10	57	139	80-140
	65	8	8	11	9	13,7	19	69	66	13,8	16	7	8	9,3	17	71,1	60	8,8	9	9	10	10	10	58	198	120-240

- **Experience Abroad**: 1 year at **NOKIA Bell Labs** (USA), from April 2016 till March 2017, under the supervision of Marina Thottan and Catello Di Martino
  - **Resilience assessment of the Software Defined Network platforms**

Telco Cloud

- Telco Cloud is meant to provide a dedicated **cloud computing solution for network operators**
  - Shifting from dedicated legacy hardware into virtualized software components
  - Delivering Telco applications that demand low latency and high throughput
  - Deploying state-of-the-art data center components that support open and interoperable cloud solutions
  - Utilizing fully IT compliant hardware able to run the most common IT cloud applications in parallel with Telco

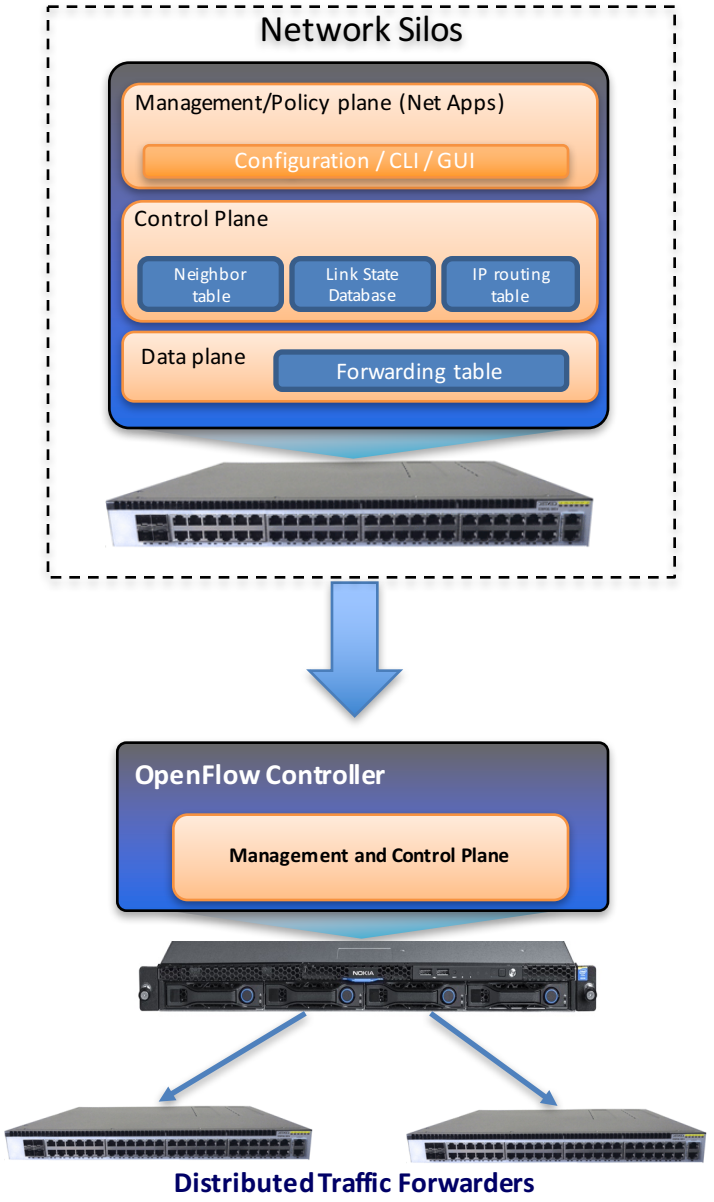
- Traditional Networking is not the proper solution for Telco Carrier Clouds
  - Closed equipments and over specified
  - Innovation driven by equipment vendor
  - Not designed for virtualization and On-Demand provisioning
  - Not cost effective (CAPEX & OPEX)

Software Defined Networking

- The current design for network equipment is **monolithic/tightly integrated**
  - Management plane / configuration
  - Custom Control plane / Decision
  - Data plane / Forwarding

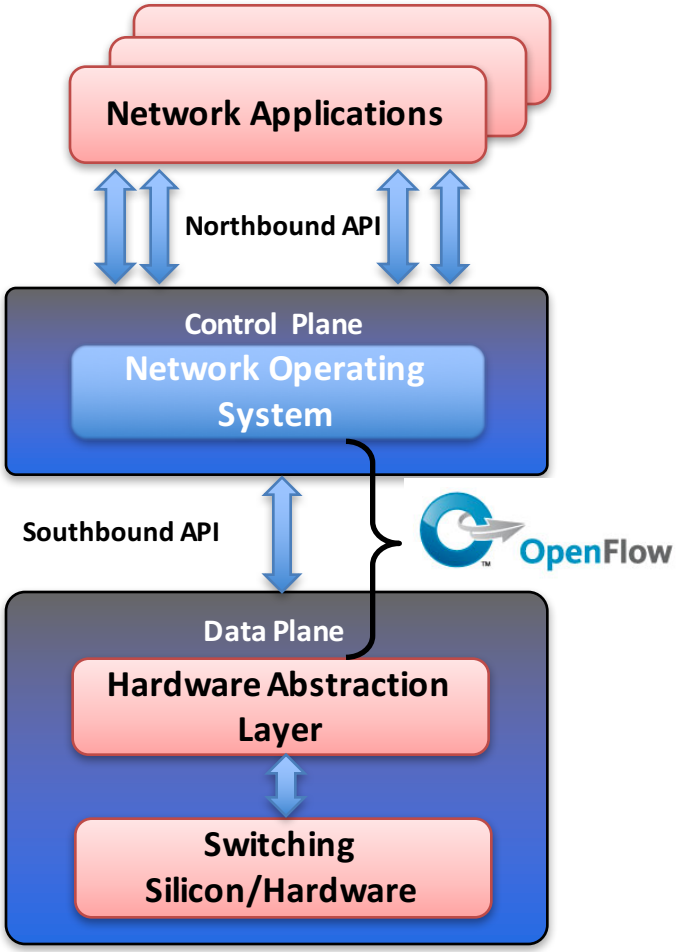
- **Software defined networking** (SDN) is an approach for designing, building and managing **computer networks that separate and abstract elements of these systems**

- In the SDN paradigm, the processing does not happen in the same device



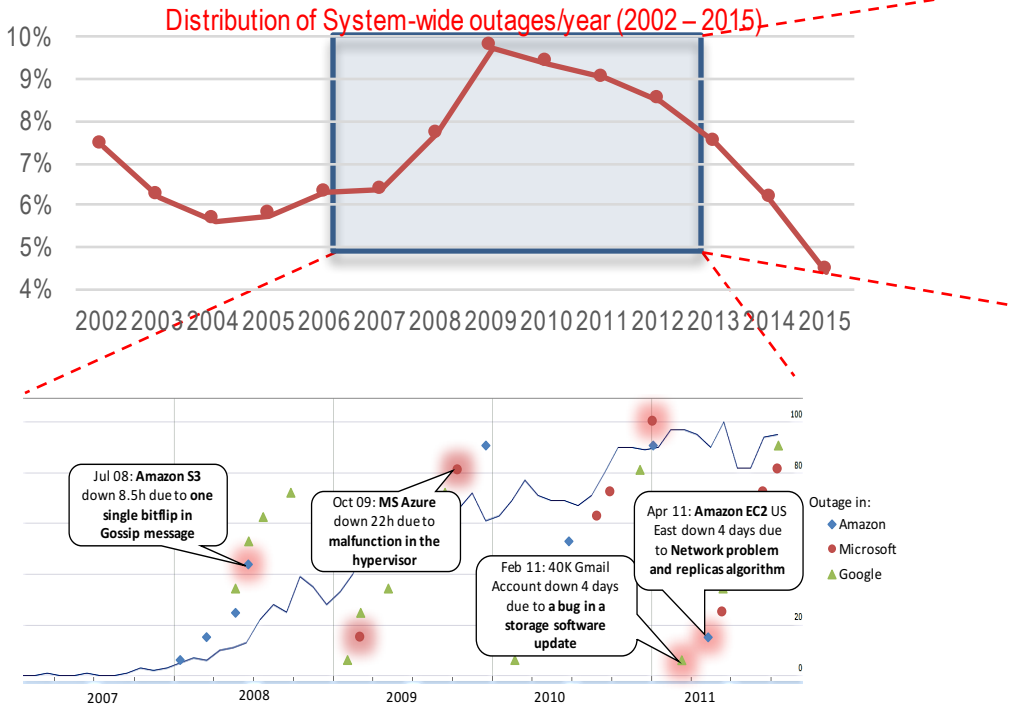
SDN Key Principles

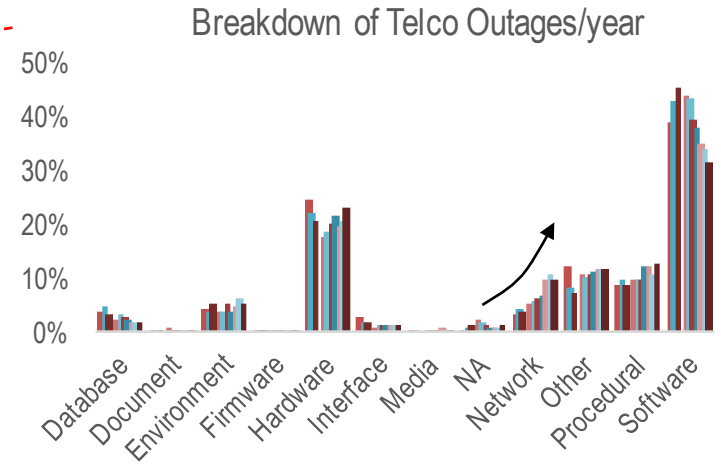
- **Logically Centralized Intelligence**
  - SDN separates network entities into **Control Plane** (the brain) **and Data Plane** (the muscles) network entities
    - The SDN Controller offers a logically centralized view of the network
    - The underlying network infrastructure is abstracted for applications and network services
- **Programmability**
  - The network control becomes directly programmable
  - A central entities maintains, controls and programs the data plane state
- **Higher Abstraction**
  - SDN decouples form specific networking hardware, from business applications, and from control plane and virtualized configuration



Threats to Resilient Carrier Grade SDNs

- The **"softwarization"** approach has its drawback
  - Software is the **major** cause of System outages
    - **Short-term** service delivery does not allow extensive testing
    - Majority of outages are due to the **inability** of failovers software to execute as expected
- **High resiliency** is the answer





Failures are **unavoidable**

Amazon EC2 4 days East Cost outage in because of network failures
Impact: several million $ refund to customers of the East Coast availability region

LTE data outages caused by core service delivery network failures
Impact: 20+ millions users from California to Maryland unable to get LTE services for 24h

Streaming Service Hit by 8h outage triggered by failure of the network failover
Impact: 20,000,000 users affected

Open Challenges (1/2)

- SDNs technologies have raised new challenges of achieving **network resiliency**

  - **Due to the decoupling between control and data plane**
    - Network resilience depending on the fault-tolerance of data-plane and on the high availability of the (logically) centralized control functions
    - The latency can affect the communication between the data and control plane

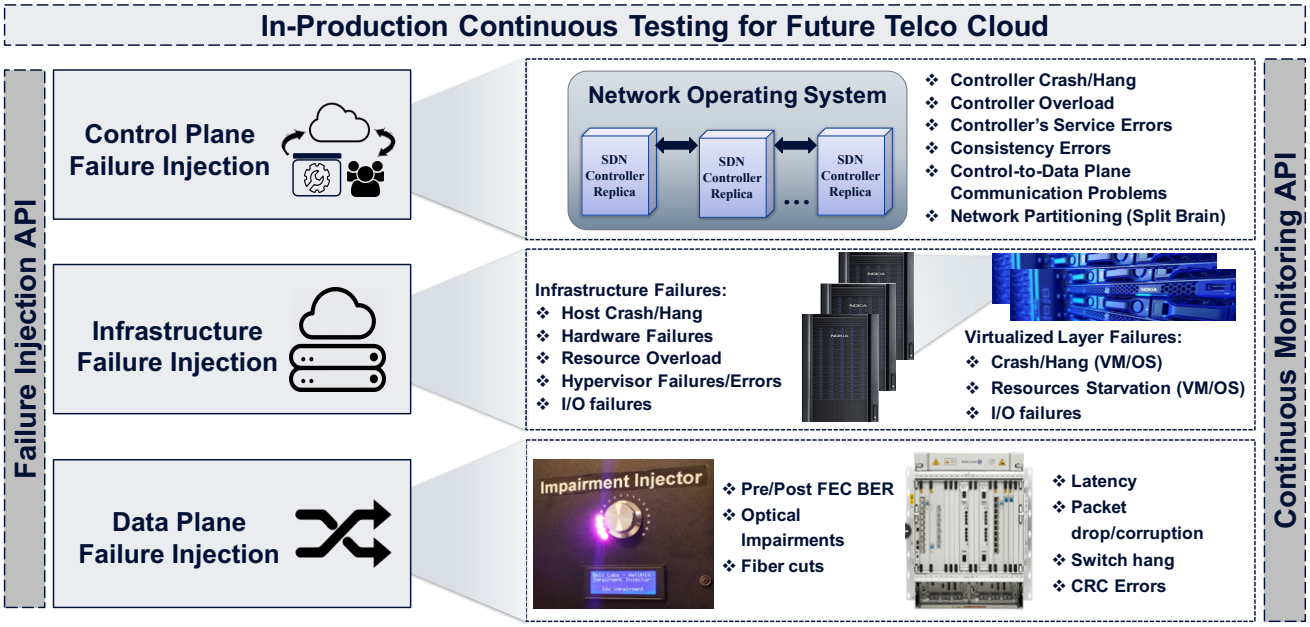
  - **Due to the natural implementation of SDNs as distributed systems**
    - The SDNs can suffer of performance loss due to the unbalance of the network resource management
    - The network latency can affect the synchronization process between the SDN controllers The partitioning of the SDN cluster can lead to network partitioning and inconsistent global network state

  - **Due to the adoption of technologies to virtualize the network services**
    - Failures periodically occur in data center and cloud ecosystems

Open Challenges (2/2)

- There is the need of a methodology and tools to:
  - **Characterize the resiliency** and **reliability** of an SDN platform
    - ✓ **RQ1: Is the SDN controller resilient to failure scenarios?**
    - ✓ **RQ2: Is the SDN distributed platform resilient to controller(s) failures?**
    - ✓ **RQ3: How the SDNs performance and availability are affected by faulty conditions?**
  - Assess the **efficacy** and **effectiveness** of the SDN resilience mechanisms
    - ✓ **RQ4: Are the SDNs resilience mechanisms efficient enough to meet the 5 nines availability requirement of Carrier Network?**
      - Efficacy of the failure detection and mitigation
      - Effectiveness of the failover and fallback functionalities
      - Ability to maintain a consistent view of the network under faulty conditions

Research Context: a Bell Labs Vision for Future Telco



- Improving the network **resilience** through **in-production continuous testing**
  - **Failure injection** as a mean to generate ground truth on
    - Effectiveness of detection mechanisms (How well does the system detect the failure?)
    - Effectiveness of recovery (How well does the system react to a failure?)
    - Manifested failures (What did it happen?)
  - **Continuous feedback** on the system resilience and reliability
    - On-line improvement of the failure detectors and failover
    - Observing the system behavior under know faulty conditions
  - Failure injected at different layers of the Telco infrastructure
    - **Data Plane**: emulate faulty network appliances
    - **Infrastructure Level**: emulate faulty physical nodes or virtualized hosts
    - **Control Plane**: emulate faulty network controllers

Why Failure Injection Testing?

- Failures are **unavoidable** in complex ecosystems, such as SDNs
  - Even if a system is designed to be fault tolerant
    - Exception handling, fault tolerance and isolation, redundancy, etc.
  - … and was tested to increase the confidence on its quality
    - Unit testing, integration testing, stress/load testing, etc.
  - … testing distributed system is hard
    - Web scale traffic, complex interactions between subsystems, 3rd party services, etc.
    - **Traditional software testing techniques appear insufficient** to evaluate the resilience and availability of a distributed ecosystems

- To be **more proactive**, for ensuring the fault tolerant mechanisms **without to wait the occurrence of a failure**
  - Harnessing failure injection to ensure that the system is **fault-tolerant**
  - **Continuously** test the system ability to survivor "rarely"
  - Inject failures during **production hours**
    - To simulate failures modes that are possible in a production environment
    - To simulate failures modes under controlled circumstances
    - Exploiting real workload instead of synthetic workload

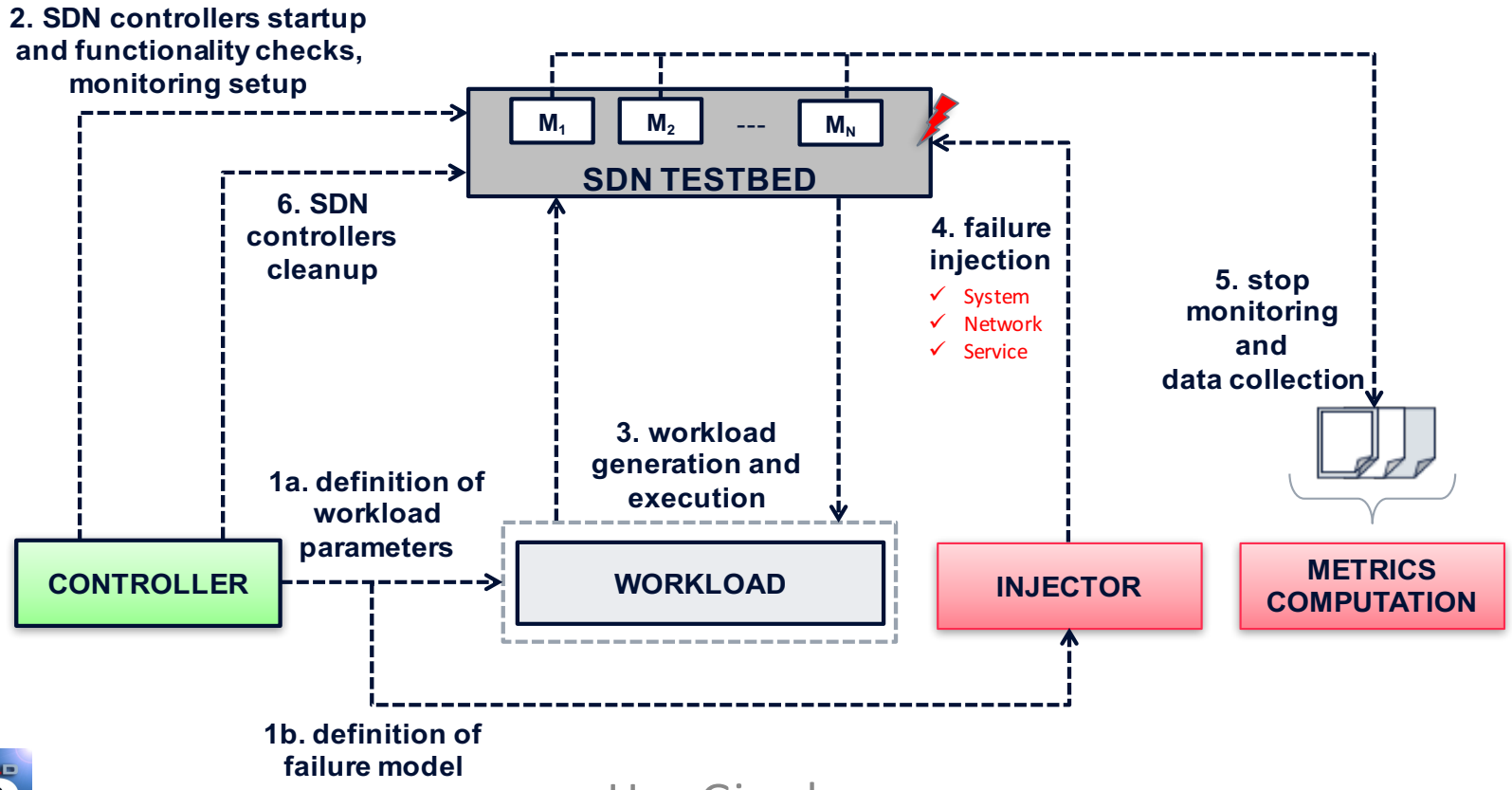
Failure Injection Testing: Netflix and others

- Failure injection techniques are widely used in both industrial and research fields
  - Netflix created **chaos monkey**
    - A wide range of tools to deliberately inject failures in their "micro-services" ecosystem
    - Aiming to assess the resilience of the distributed platform to random (chaos) failures, e.g. services crashes, corrupted communications …
  - Along this line they proposed the **chaos engineering** principles
    - "A discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand faulty conditions in production"

- Besides Netflix, other organizations apply similar techniques to continuously test the resiliency of their services
  - Amazon and Google
    - Amazon created **GameDay**, a program designed to increase resilience by injecting major faults into critical systems
  - Microsoft
    - Microsoft exploits Chaos engineering (**Search Chaos Monkey**) to test the resilience of its Azure Search service
  - Facebook
    - Facebook continuously runs infrastructure stress test to assess the resilience of the overall system beyond their platform (**Facebook Stress Test**)

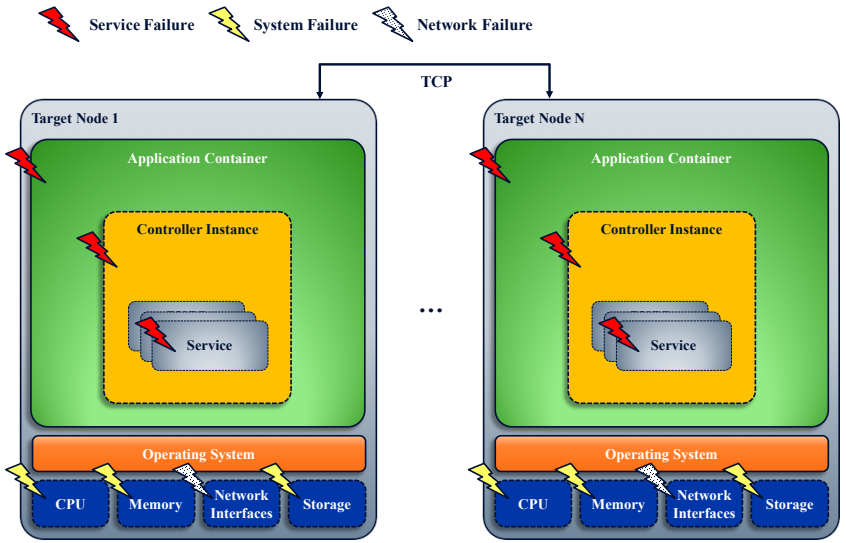
SDN resilience assessment: a failure injection methodology

- We propose a **methodology** and **infrastructure** for continuous testing in carrier-grade SDNs by an **automatic** and **configurable failure injection framework**
  - The approach aims at assessing the effectiveness of the SDNs resilience mechanisms
  - enables to inject failure in-production environment
  - provides feedback to propose improvement for the resilience of the nowadays SDNs



The Failure Model

- The failures can be set as:
  - **Transient**: injected only once and removed after a specified amount of time (emulate **temporary** failure)
  - **Intermittent**: periodically injected and left in the system for a specified amount of time (emulate **temporary but recurrent** faulty conditions)
  - **Permanent**: injected and never removed from the system (emulate **persistent** faulty conditions)



- We use failure modes that encompass most of the **common failure observer in distributed systems**
  1. **System Failures**: affecting the computational resources as well as I/O operations of a target node.

Failure class	Failure type	Description
System Failures	**System Hang**	Stuck the system on an infinite loop without releasing the resources. The system's network interfaces are still responding.
	System Starvation	Cause the starvation of all the available system resources. The system first slow down, then crashes.
	System Outage	Cause a fatal error from which the system cannot safely recover (e.g. kernel panic).
	System Shutdown	Gracefully shut down the system.
	CPU Shutdown	Restrict the number of available CPUs.
	Disk Saturation	Cause the saturation of the disk, mimicking disk full errors.
	Memory Saturation	Cause the saturation of the memory, mimicking memory full errors.
	Burn CPU	Spawn CPU-bound processes, mimicking a faulty CPU and/or noisy process.
	Burn I/O	Spawn I/O-bound processes, mimicking a faulty disk and/or noisy process.

The Failure Model

2. **Network Failures**: affecting the communication between SDN controllers, and the data-control plane communication

Failure class	Failure type	Description
Network Failures	**Black-hole**	Abruptly drop the network communications towards a specific address or a subset of addresses.
	Packet Reject	Abruptly drop the inbound and/or outbound packets sent to specified address and/or port.
	Packet Drop	Quietly drop the inbound and/or outbound packets sent to specified address and/or port.
	Packet Latency	Induce artificial delays for packets sent to specified address and/or port.
	Packet Loss	Induce artificial losses for the packets sent to specified address and/or port.
	Packet Re-order	Induce a mis-ordering of certain packets sent to specified addresses and/or port.
	Packet Duplication	Induce duplication of certain packets sent to specified addresses and/or port.
	Packet Corruption	Induce a random noise by introducing an error in a random position of certain packets sent to specified addresses and/or port.
	Throttling	Induce a bandwidth limitation to the outgoing network traffic with specified addresses and/or port.

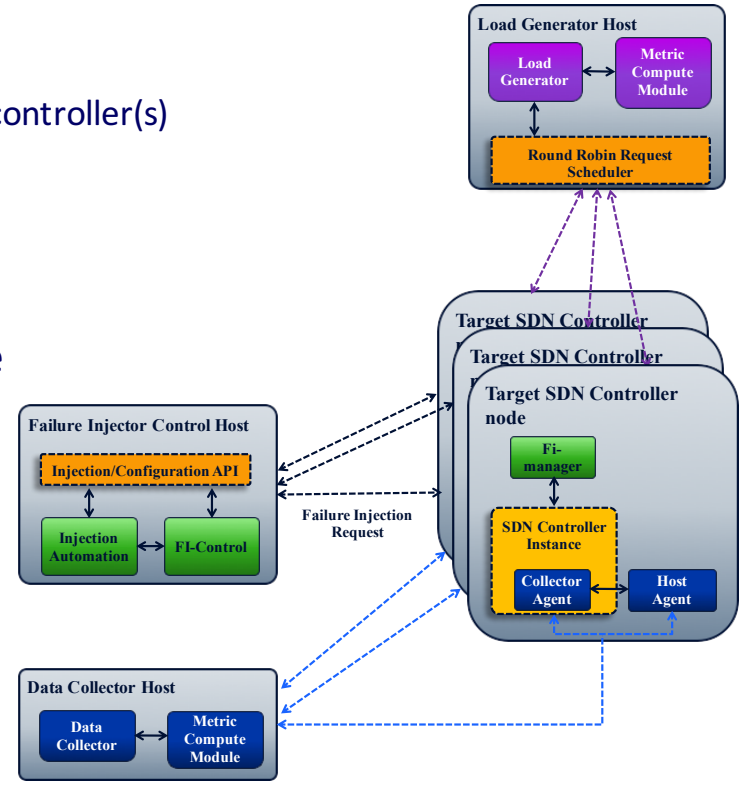
3. **Service Failures**: affecting the SDN controller services

Failure class	Failure type	Description
Service Failures	**Kill Process**	Quietly terminate the controller process (*SIGTERM* signal), mimicking a faulty service.
	Process Corruption	Randomly corrupt the state of the SDN controller process, mimicking a service misbehaviour.
	Controller Stop	Quietly stop of the SDN controller process .
	Controller Restart	Gracefully restart the SDN controller.
	Dependency Stop	Quietly stop one or more dependencies, i.e. modules, of the SDN controller.

A failure injection infrastructure (1/3)

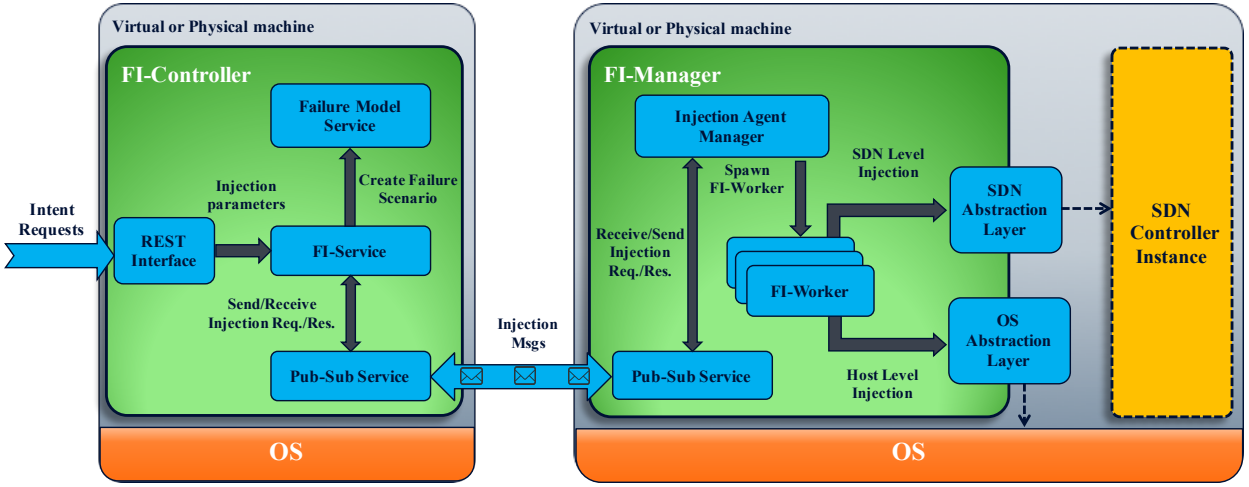
- The infrastructure encompasses three main components, each in turn composed of a set of interoperating sub components:

1. The **Workload Generator** produces the workload for the SUT
   - ➢ It is a a **standalone benchmarking tool** for the Northbound Interfaces of the SDN controllers
     - Offline and on-line configurability
     - Providing high level performance measurements of the SDN controller(s)
       - ➢ Intent/Flow request throughout
       - ➢ Intent/Flow request latency
       - ➢ System availability
   - ➢ Reproduces policy-based application requests, the **Intent Requests**
     - An Intent is an immutable request model specifying the requirements of an application's demand, which specify how the network must be programmed at low level
   - ➢ Equally distributes the requests towards the northbound API of the deployed controllers



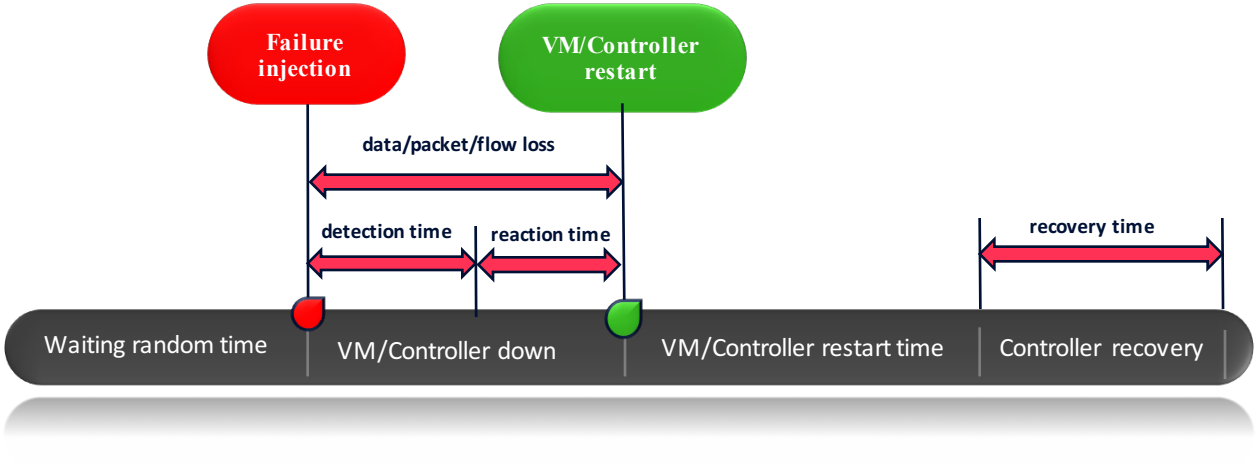
A failure injection infrastructure (2/3)

2. The **Failure Injector** is an extensible ready-to-use component which can be integrated seamlessly into the target system to inject multiple failure modes. It includes:
   - A **failure injection controller**:
     - Provides a REST interface to the end-user to define the failure injection parameters
     - Translates user requests into corresponding request for the injection manager
     - Provides an internal engine to automate the failure injection experiments and data collection
   - A **failure injection manager** which is responsible to actually perform the injection on the target SDN controller

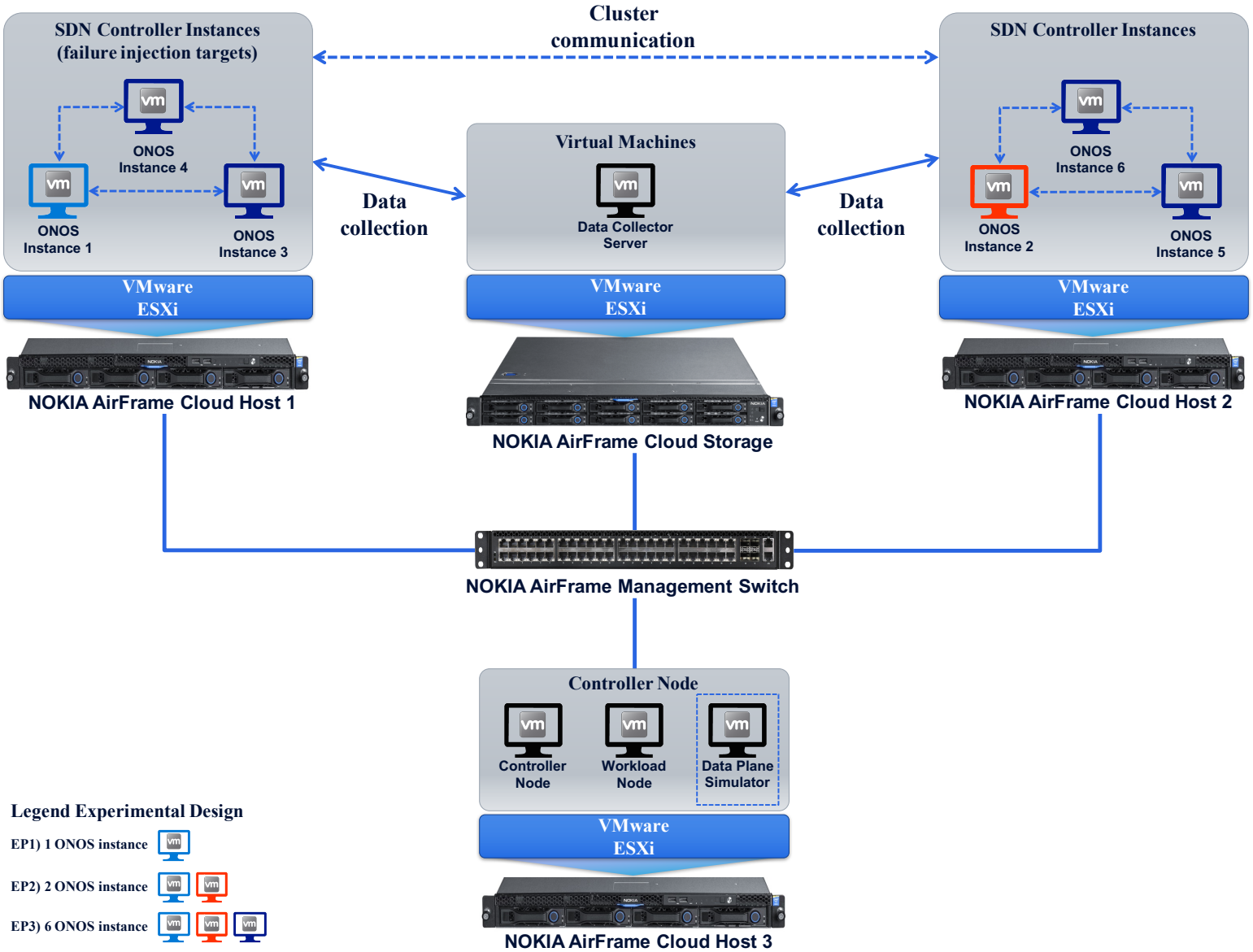


A failure injection infrastructure (3/3)

3. The **Data Collector** is in charge of collecting the data concerning to the system under consideration
   - Listen to and reporting the events occurring in the SDN ecosystem
   - Collecting metrics concerning the resource consumption of the SDN controller hosting machines
   - Allows the collection of metrics exploitable to compute some basic measurements of the effectiveness and efficacy of the system resilience mechanisms
     - **Recovery time** (time the controller needs to synchronize its status with the other controllers and the data-plane)
     - **Detection latency** (time needed until a faulty condition is detected by the controllers)
     - **Reaction time** (the time needed to activate the failover of a faulty controller)



The testbed


SDN Controller Instances
(failure injection targets)
ONOS
Instance 4
ONOS
Instance 1
ONOS
Instance 3
Cluster
communication
SDN Controller Instances
ONOS
Instance 6
ONOS
Instance 2
ONOS
Instance 5
Data
collection
Virtual Machines
Data Collector
Server
Data
collection
VMware
ESXi
VMware
ESXi
VMware
ESXi
NOKIA AirFrame Cloud Host 1
NOKIA AirFrame Cloud Storage
NOKIA AirFrame Cloud Host 2
NOKIA AirFrame Management Switch
Controller Node
Controller
Node
Workload
Node
Data Plane
Simulator
VMware
ESXi
NOKIA AirFrame Cloud Host 3
Legend Experimental Design
EP1) 1 ONOS instance
EP2) 2 ONOS instance
EP3) 6 ONOS instance

Ongoing activities

- Work in progress
  - Experimental evaluation of the Intent Based Networking (IBN) framework provided by Open Network Operating System (ONOS) and Open Daylight (ODL)
  - Experimental evaluation of the resilience and a reliability of ONOS and ODL under faulty conditions

- What's next
  - Extend the failure model with further failure scenarios
  - Propose possible performance improvement for the IBN framework
  - Propose optimization of the resilience mechanisms provided by the SDN controllers

Publications and References

[1] Carrozza, G., Cinque, M., Giordano, U., Pietrantuono, P. & Russo, S. (2015). Prioritizing correction of static analysis infringements for cost-effective code sanitization. In Proceedings of the 2th International Workshop on Software Engineering Research and Industrial Practices (SER&IPs 2015). 37th "International Conference on Software Engineering" (ICSE 2015), Florence, May 16th to 24th.

[2] Diego Kreutz, Fernando M. V. Ramos, Paulo E. Verissimo, Christian E. Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. Pro- ceedings of the IEEE, 103(1):14–76, 2015.

[3] BrunoA.A.Nunes,MarcMendonca,Xuan-NamNguyen,KatiaObraczka,andThierryTurletti. A survey of Software-Defined Networking: Past, Present, and Future of Programmable Net- works. IEEE Communications Surveys Tutorials, 16(3):1617–1634, 2014.

[4] Roberto Natella, Domenico Cotroneo, and Henrique S. Madeira. Assessing Dependability with Software Fault Injection: A Survey. ACM Computing Surveys, 48(3):44:1–44:55, 2016.

[5] ONOS Project. ONOS White Paper. http://onosproject.org/wp-content/uploads/2014/11/ Whitepaper-ONOS-final.pdf (accessed January 2017), 2014.

[6] Catello Di Martino, Veena Mendiratta, and Marina Thottan. Resiliency Challenges in Ac- celerating Carrier-Grade Networks with SDN. In Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pages 242–245. IEEE, 2016.

[7] Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali. On Scalability of Software- Defined Networking. IEEE Communications Magazine, 51(2):136–141, 2013.

[8] SethGilbertandNancyLynch.Brewer'sConjectureandtheFeasibilityofConsistent,Available, Partition-Tolerant Web Services. ACM SIGACT News, 33(2), 2002.

[9] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Sub- baiah Venkata, Jim Wanderer, Junlan Zhou, and Min Zhu. B4: Experience with a globally- deployed software defined WAN. ACM SIGCOMM Computer Communication Review, 43(4):3–14, 2013.

[10] Song Huang, Zhiang Deng, and Song Fu. Quantifying entity criticality for fault impact analysis and dependability enhancement in software-defined networks. In Proceedings of the IEEE 35th International Performance Computing and Communications Conference (IPCCC), pages 1–8.

[11] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos Engineering. IEEE Software, 33(3):35–41, 2016.

Thanks!